**CSCI 3353 Object-Oriented Design**
Homework Assignment 2
Due Monday, February 5

Consider the following dice game:

> **Players:** Two or more players take turns.
> **Object:** To be first person to either accumulate 30 chips.
> **Setup:** The pot is initially empty and all players start with no chips. New chips will be added to the pot from an inexhaustible bank.
>
> When your turn begins, add one chip from the bank to the pot. Roll a four-sided die. If you get a 1, you have "aced out", and your turn ends. Otherwise, you may either take the pot or keep going. If you keep going, add two more chips to the pot and roll two dice. If you roll a 1 on either die, you have aced out. Otherwise, you may take the pot or keep going, this time adding three chips and rolling three dice. Continue until you either ace out or decide to take the pot.

Your task is to write a Java program to play the above game. The main class should be named *HW2DiceGame*.

The first thing the program should do is let the user specify the players. The user should be able to choose how many players are in the game, and choose the type of each player. There should be several player types to choose from: a human player, and several kinds of computer players. There is no restriction on how many players of a given type can be playing a game, and what their positions are. In particular, it should be possible to run a game that has no humans, all humans, or any combination thereof.

The difference between a human player and a computer player is in how their moves are transmitted to the game. A human enters a move via the keyboard, whereas a computer calculates its move. In either case, a player will most likely determine its move based on the state of the game (the number of chips in the pot, the number of chips held by each player, etc.).

Your code should supply two kinds of computer players, having the following move-calculation algorithms:

- A timid player, who cashes out as soon as possible.
- A crafty player, who will not cash out until it has more chips than anyone else.

In addition to determining the winner, your program should print the turn-by-turn action and the state of the game after each turn. Here is an example of the output of the beginning of a game between two players. (Note that I give each player a name. You can do what you want.)

```
sam stopped after 1 rolls and won 1 chips
Chip count: pat:0 sam:1
The pot contains 0 chips.

pat aced out after 2 rolls
Chip count: pat:0 sam:1
The pot contains 3 chips.

sam stopped after 2 rolls and won 6 chips
Chip count: pat:0 sam:7
The pot contains 0 chips.
```

Your job is to write this program in the best object-oriented way that you can. You can organize your program into whatever classes you wish, having whatever methods you deem appropriate.

There is one restriction: The three kinds of player must be implemented by the classes *HumanPlayer*, *TimidPlayer*, and *CraftyPlayer*, each of which implements the interface *Player*. You will discover that the player classes have a lot of code in common; that's ok. In Chapter 3 we will see how to use an abstract class to eliminate the duplicate code. However, I DO NOT want you to do that in this assignment.

SUBMISSION GUIDELINES:

Your code should consist of the main file *HW2DiceGame.java*, the interface *Player.java*, files for the three player classes, and any other classes you wish to create. All files should be in the package *hw2*. Submit each file individually to Canvas; do NOT zip them. You should also submit a document that explains the design decisions you made and why you made them. Your document should include a class diagram.

SOME HINTS:

As you try to decide what methods the player classes should have, as well as what other classes you might want, think about what information a potential class will manage. In general, a class is characterized by its field values; it hides those values and provides public methods based on those values. In effect, you can think of that class as the "manager" of those values. (Our *BankClient*, *Bank*, and *BankAccount* demo classes are all like this.) A class that doesn't manage any data isn't very useful.

If you get stuck (and you probably will), just start writing code for the classes you have. As you write, you might notice places where another class would be useful. When you eventually get your code to work, you can always go back and refactor it to be more object-oriented, as in HW 1.

I want to stress is that this is an iterative process. Your first version of the program will probably not be the best. You need to start early, to give yourself time to create a second (or third or fourth) version.